TENSOR CORE PROGRAMMABILITY AND PROFILING FOR AI AND HPC APPLICATIONS

Griffin Lacey
Max Katz
• Tensor Cores enable fast mixed precision matrix multiplications
• Growing number of AI/HPC examples accelerated up to 25x
• Mature software support with high-level APIs and Nsight developer tools
• All you need is Volta / Turing GPU
OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores
OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores
FIRST, WHAT IS PRECISION?

- Precision is a measure of numerical detail
- Floating Point (FP) is a representation of real numbers supporting the tradeoff of:
  - Precision (significand)
  - Range (exponent)
- Lower precision numbers have computational performance advantages

Figure: https://devblogs.nvidia.com/tensor-cores-mixed-precision-scientific-computing/
WHAT ARE TENSOR / CUDA CORES?

Figures: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
VOLTA GV100 SM

**GV100**

- **FP32 units**: 64
- **FP64 units**: 32
- **INT32 units**: 64
- **Tensor Cores**: 8
- **Register File**: 256 KB
- **Unified L1/Shared memory**: 128 KB
- **Active Threads**: 2048

CUDA CORES
VOLTA TENSOR CORE
Half/Mixed Precision 4x4 Matrix Multiply-Accumulate

\[ D = AB + C \]

FP16 or FP32

Turing Tensor Cores: support for int8, int4
VOLTA TENSOR CORE

Full Warp 16x16 Matrix Math

Warp-synchronizing operation for cooperative matrix math

Aggregate Matrix Multiply and Accumulate for 16x16 matrices

Result distributed across warp
OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores
TENSOR CORES FOR AI

• Simple trick for 2x to 5x faster deep learning training
  • Accomplished in few lines of code
  • Models can use same hyperparameters
  • Models converge to same accuracy
• Half the memory traffic and storage enabling larger batch sizes
• AI community is trending towards low precision as common practice
HOW TO USE TENSOR CORES

- Exposed as instructions in CUDA under WMMA API (Warp Matrix Multiply Accumulate)
- Used by cuDNN, cuBLAS, CUTLASS to accelerate matrix multiplications and convolution
- Tensor Core kernels used implicitly on FP16 ops from DL frameworks PyTorch / TensorFlow / etc...
- High-level tools (e.g. PyTorch Apex) convert everything automatically and safely
MIXED PRECISION TRAINING

1. Model conversion
2. Master weight copy
3. Loss scaling
1. MODEL CONVERSION

- Make simple type updates to each layer:
  - Use FP16 values for the weights and inputs

# PyTorch
layer = torch.nn.Linear(in_dim, out_dim).half()

# TensorFlow
layer = tf.layers.dense(tf.cast(inputs, tf.float16), out_dim)
2. MASTER WEIGHTS

- FP16 alone is sufficient for some networks but not others; keep FP32 copy of weights

```python
param = torch.cuda.FloatTensor([1.0])
print(param + 0.0001)  # 1.0001

param = torch.cuda.HalfTensor([1.0])
print(param + 0.0001)  # 1
```

When update/param < 2^-11, updates have no effect.
3. LOSS SCALING

• Range representable in FP16: ~40 powers of 2

• Gradients are small:
  • Some lost to zero
  • While ~15 powers of 2 remain unused

• Loss scaling:
  • Multiply loss by a constant $S$
  • All gradients scaled up by $S$ (chain rule)
  • Unsacle weight gradient (in FP32) before weight update
MIXED PRECISION TRAINING

1. Model conversion
2. Master weight copy
3. Loss scaling

Automated Mixed Precision (AMP) (e.g. PyTorch Apex)
N, D_in, D_out = 64, 1024, 512
x = Variable(torch.randn(N, D_in)).cuda()
y = Variable(torch.randn(N, D_out)).cuda()

model = torch.nn.Linear(D_in, D_out).cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    optimizer.zero_grad()

    loss.backward()
    optimizer.step()
**PYTORCH APEX AMP 1.0**

\[ N, D_{in}, D_{out} = 64, 1024, 512 \]

\[ x = \text{Variable(torch.randn}(N, D_{in})).cuda() \]
\[ y = \text{Variable(torch.randn}(N, D_{out})).cuda() \]

```python
model = torch.nn.Linear(D_in, D_out).cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
model, optimizer = amp.initialize(model, optimizer, opt_level="O1")

for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    optimizer.zero_grad()
    with amp.scale_loss(loss, optimizer) as scaled_loss:
        scaled_loss.backward()
    optimizer.step()
```
# MIXED PRECISION SPEEDUPS
Not Limited to Image Classification

<table>
<thead>
<tr>
<th>Model</th>
<th>FP32 -&gt; FP16 Speedup</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNMT (Translation)</td>
<td>2.3x</td>
<td>Iso-batch size</td>
</tr>
<tr>
<td>FairSeq Transformer (Translation)</td>
<td>2.9x, 4.9x</td>
<td>Iso-batch size 2x lr + larger batch</td>
</tr>
<tr>
<td>ConvSeq2Seq (Translation)</td>
<td>2.5x</td>
<td>2x batch size</td>
</tr>
<tr>
<td>Deep Speech 2 (Speech recognition)</td>
<td>4.5x</td>
<td>Larger batch</td>
</tr>
<tr>
<td>wav2letter (Speech recognition)</td>
<td>3.0x</td>
<td>2x batch size</td>
</tr>
<tr>
<td>Nvidia Sentiment (Language modeling)</td>
<td>4.0x</td>
<td>Larger batch</td>
</tr>
</tbody>
</table>
OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores
TENSOR CORES FOR HPC

- Mixed precision algorithms are **increasingly popular**
  - It is common to combine **double + single precision**, or **floating point + integer**
- Similar to AI:
  - Use low precision to **reduce memory traffic and storage**
  - Use Tensor Core instructions for **large speedups**
Researchers from ICL/UTK
Accelerated FP64 LU factorization 4x using Tensor Cores in MAGMA
Compute initial solution in FP16, then iteratively refine solution
Achieved FP64 TFLOPS: 5.8
Achieved FP16->FP64 TFLOPS: 24

Data courtesy of: Azzam Haidar, Stan. Tomov & Jack Dongarra, Innovative Computing Laboratory, University of Tennessee
"Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers", A. Haidar, S. Tomov, J. Dongarra, N. Higham SC‘18
GTC 2018 Poster P8237: Harnessing GPU’s Tensor Cores Fast FP16 Arithmetic to Speedup Mixed-Precision Iterative Refinement Solves
EARTHQUAKE SIMULATION

- Researchers from University of Tokyo, Oak Ridge National Laboratory (ORNL), and the Swiss National Supercomputing Centre
- Solver called MOTHRA achieved 25x compared to standard solver
- Used AI to identify where to apply low or high precision in solver
- Used a combination FP64, FP32, FP21 and FP16 to further reduce computational and communication costs

OUTLINE

1. What are Tensor Cores?
2. Tensor Cores for AI
3. Tensor Cores for HPC
4. Profiling Tensor Cores
NSIGHT DEVELOPER TOOLS
NSIGHT PRODUCT FAMILY

Standalone Performance Tools

Nsight Systems - System-wide application algorithm tuning
Nsight Compute - Debug CUDA API and optimize CUDA kernels
Nsight Graphics - Debug/optimize specific graphics apps

IDE Plugins

Nsight Eclipse Edition/Visual Studio - editor, debugger, some perf analysis
NSIGHT SYSTEMS
Next-Gen System Profiling Tool

System-wide application algorithm tuning
Multi-process tree support

Locate optimization opportunities
Visualize millions of events on a fast GUI timeline
Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs
CPU algorithms, utilization, and thread state
GPU streams, kernels, memory transfers, etc

Multi-platform: Linux & Windows, x86-64 & Tegra, MacOSX (host only)
NSIGHT COMPUTE
Next-Gen Kernel Profiling Tool

Key Features:
• Interactive CUDA API debugging and kernel profiling
• Fast Data Collection
• Improved Workflow (diffing results)
• Fully Customizable (programmable UI/Rules)
• Command Line, Standalone, IDE Integration

OS: Linux, Windows, ARM, MacOSX (host only)
GPUs: Pascal (GP10x), Volta, Turing
USING NSIGHT SYSTEMS
COLLECT A PROFILE WITH NSIGHT SYSTEMS

$ nsys profile /usr/bin/python train.py

Generated file: report.qdrep

Import for viewing into the Nsight Systems UI

The Nsight Systems UI can also be used for interactive system profiling
LOCATING TENSOR CORE KERNELS

On Volta V100, CUDA kernels using tensor cores contain the string “s884”

Examples:

volta_fp16_s884gemm_fp16_128x64_ldg8_f2f_nn
volta_fp16_s884cudnn_fp16_256x64_ldg8_relu_f2f_exp_interior_nhwc2nchw_tn_v1

These are kernels with HMMA (half-precision matrix multiply and accumulate) machine instructions
COMING SOON: SQLITE DATABASE EXPORT
USE NSYS-EXPORTER TO CREATE SQLITE DB

nsys-exporter -s report.qdrep

Generated DB: report.sqlite

Interact with this like any SQLite database
ASSOCIATE KERNEL NAMES WITH EVENTS

ALTER TABLE CUPTI_ACTIVITY_KIND_RUNTIME ADD COLUMN name TEXT;
UPDATE CUPTI_ACTIVITY_KIND_RUNTIME SET name =
    (SELECT value FROM StringIds
     JOIN CUPTI_ACTIVITY_KIND_KERNEL AS cuda_gpu
        ON cuda_gpu.demangledName = StringIds.id
     AND CUPTI_ACTIVITY_KIND_RUNTIME.correlationId = cuda_gpu.correlationId);
LOCATE KERNELS USING TENSOR CORES

SELECT *

FROM

CUPTI_ACTIVITY_KIND_RUNTIME as cupti

WHERE

cupti.name LIKE '%s884%'
USING NSIGHT COMPUTE
KERNEL PROFILES WITH NSIGHT COMPUTE

$ nv-nsight-cu-cli /usr/bin/python train.py

This is expensive for a typical DL training session because it will collect metrics for every kernel; consider profiling fewer kernels.

For example, to profile *s884* kernels on all streams, but only on the fifth invocation:

$ nv-nsight-cu-cli --kernel-id ::s884:5 /usr/bin/python train.py

The Nsight Systems UI can also be used for interactive kernel profiling
INTERLUDE: FINDING TENSOR CORE METRICS

Isolating which GPU metrics measure tensor cores:

$ nv-nsight-cu-cli --devices 0 --query-metrics | grep -i tensor

... 

smsp__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active
sm__pipe_tensor_op_hmma_cycles_active.avg
sm__inst_executed_pipe_tensor.avg.per_second

...
TENSOR CORE PERFORMANCE

Now add in what we know about which metrics to looks for:

$ nv-nsight-cu-cli --kernel-id ::s884:5 --metrics
smsp_pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active
/usr/bin/python train.py

volta_s884cudnn_fpp16_128x128_ldg8_wgrad_idx_exp_interior_nhwc_nt, 2019-Mar-17 02:54:29, Context 1, Stream 23

Section: Command line profiler metrics

--------------------------------------------------------------------- %
smsp_pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active  79.03
---------------------------------------------------------------------
NSIGHT COMPUTE UI
SUMMARY
NVIDIA TOOLS FOR TENSOR CORE PROFILING

System, application, and kernel level profiling solutions

- **Nsight Systems**: high-level application view; locate kernels that used tensor cores
- **Nsight Compute**: drill down into specific kernels for detailed performance analysis
- Starting with version 19.03, [NVIDIA GPU Cloud](https://ngc.nvidia.com) (NGC) optimized deep learning containers package Nsight Systems and Nsight Compute. Download and try it now!
DEVELOPER TOOLS AT GTC19

Talks:
S9751: Accelerate Your CUDA Development with Latest Debugging and Code Analysis Developer Tools, Tue @9am
S9866 - Optimizing Facebook AI Workloads for NVIDIA GPUs, Tue @9am
S9345: CUDA Kernel Profiling using NVIDIA Nsight Compute, Tue @1pm
S9661: Nsight Graphics - DXR/Vulkan Profiling/Vulkan Raytracing, Wed @10am
S9503: Using Nsight Tools to Optimize the NAMD Molecular Dynamics Simulation Program, Wed @1pm

Hands-on labs:
L9102: Jetson Developer Tools Training Lab, Mon @9am, 11:30am
L9124: Debugging and optimizing CUDA applications with Nsight products on Linux training lab, Tue @8am, 10am

Connect with the Experts (where DevTools will be available):
CE9123: CUDA & Graphics Developer Tools, Tue @2pm, Wed @3pm
CE9137: Jetson Embedded Platform, Tue @12pm, 5pm, Wed @1pm, 4pm, Thu @12pm

Podium: Demos of DevTools products on Linux, DRIVE AGX & Jetson AGX at the showfloor
   Tue @12pm - 7pm
   Wed @12pm - 7pm
   Thu @11am - 2pm